Architectural Requirements for an Open Source Component and Artefact Repository system within GENESIS

Cornelia Boldyreff, David Nutter and Stephen Rank

October 27, 2004

David.Nutterdurham.ac.uk

Abstract

When software is being created by distributed teams of software engineers, it is necessary to manage the work-flow, processes, and artefacts which are involved in the engineering process. The GENESIS project aims to address some of the technical issues involved by providing a software system to support distributed development. One of the parts of the system will be known as OSCAR, a repository for managing distributed artefacts. Artefacts can be process models, software components, design documents, or any other kind of entity associated with the software engineering process. OSCAR will be designed as a light-weight distributed system, managing the storage and access to a distributed repository of artefacts.

This paper presents and discusses the requirements for OSCAR, and suggests a possible architecture for a software system which will meet those requirements. OSCAR will be a reliable and light-weight distributed system, managing both artefacts and meta-data corresponding to the artefacts. Users of OSCAR will be able to access the distributed repository through a local interface, using the searching and indexing capabilities of the system to locate and retrieve components. OSCAR must be able to store and retrieve both artefacts and meta-data efficiently. It must be possible for OSCAR to inter-operate with existing artefact management systems (such as CVS) and to collect metrics about the contents of and accesses to the repository.

The next stage in the GENESIS project is to complete the requirements for the whole of the system (in addition to the OSCAR sub-system) and then to design the software. The software will initially be developed in a traditional closed-source fashion until the first release is finished. After the first release, the GENESIS software will become open source, and will be developed accordingly.

1 Introduction

A software artefact is an item produced by any phase of a software development process which is not limited to actual software code but include requirements,

design, quality assurance and maintenance information. Consequently the management of software artefacts is different to that of software code and changes must be made to the *software repository* to reflect this.

In particular software artefacts inherit characteristics from:

- The software process that created them,
- Preceding versions of the artefact and obviously preceding versions of the software process,
- The dependencies each artefact has upon other artefacts and
- The *Actors* (whether human or machine) who are responsible for the artefact.

Traditional software repositories are generally passive data stores designed to support structured access to information by tools and external systems. OSCAR itself must support a more aware approach to artefact management through the notion of "active" software artefacts that are aware of their own evolution and present related information upon examination by system clients.

Since GENESIS is a process-aware software engineering environment, the artefact management system must support process control software. The GENESIS system requires that it is *non-intrusive* when deployed within an organisation. Consequently the support offered by GENESIS must not force participation from the users, instead the system must monitor user activities and alter its behaviour accordingly. To assure the non-intrusivity of the finished system and to satisfy the additional requirements imposed by the eventual release of the GENESIS system as Open Source Software (OSS) OSCAR must employ supported open standards. This perhaps seems a fait accompli; after all releasing GENESIS as Open Source will effectively produce a new open "standard". This approach ignores the advantages that can be gained by both adding "weight" to existing standards and re-using pre-written Open Source components. Open standards also offer related benefits [Raymond, 1999]:

- Existing development community,
- Ease of deployment,
- Ease of extension and
- Ease of accessibility.

This paper examines the requirements for the OSCAR subsystem within GENESIS. The paper is organised as follows:

Section 1.1 discusses work related to the GENESIS project and OSCAR in particular,

Section 2 describes the requirements of the domain that OSCAR operates in

Section 3 describes a sample use case for OSCAR,

Section 4 describes the high-level requirements of the OSCAR subsystem,

Section 5 describes a proposed architecture for OSCAR and suggests some design features of OSCAR,

Section 6 describes an ontology of artefacts for the OSCAR subsystem,

Section 7 outlines the development and exploitation strategy for OSCAR. Finally,

Section 8 summarises the previous section

1.1 Related Work

Partial solutions for software artefact management exist, where items produced in the software process are treated as more than mere version-controlled files. The products in this area are of three major types: commercial software, Open Source or Free software and research prototypes. In addition, these packages are primarily focused on other domains (such as collaborative working) than artefact management but possess additional artefact management capabilities.

Of the commercial packages, advanced Software Configuration Management (SCM) environments such as Starbase and Clearcase provide the closest functionality to true Software Artefact Management systems [Leblang, 1994]. Aside from an enriched file-system that provides a global name-space which artefacts inhabit, such systems also have process support in the style of traditional workflow systems.

Such systems are characterised by their heavyweight approach (process-centric) to software development and centralised architecture. For projects heavily distributed in both time and space and without access to high-availability services at all points, such an architecture is both difficult to implement, untrustworthy and potentially harmful to existing successful organisational processes [Sachs, 1995]. Careful identification of requirements and design [Selvin, 1999] will produce a system that facilitates existing organisational processes rather than prescribing new ones [Flores et al., 1988].

Open Source and research systems are generally more lightweight and specialised in their approach, focusing on one aspect of the domain such as version control, process control, collaborative working through hypermedia and so forth. Examples include the Oz and OzWeb (described in [Kaiser, 1998], [Gail E. Kaiser and Yang, 1997] and [Jiang et al., 1997]) environments. Haake [Haake, 1999] made a compelling case for this lightweight approach, arguing that flexibility of working practice if not process was more beneficial than enforcing a prescriptive focus through CSCW tools. Kukkonen has produced work in a similar vein. [Oinas-Kukkonen and Rossi, 1999]

2 Domain

The OSCAR tool will be situated in the domain of distributed software engineering. One of the key characteristics of this domain is that the users of the tool (programmers, configuration managers, requirements engineers, *etc.*) are distributed. That is, users are not necessarily all located at one physical site. Thus OSCAR must be able to manage the necessary distribution of artefacts

to users as required; it is not possible to assume that each user will have local access to the repository.

Entities in the domain are:

Actors In general, a user is a software engineer of some kind (requirements engineer, software designer, tester, *etc.*), a person with managerial responsibility for the software process or indeed a software agent acting on behalf of these other Actors.

Artefacts Artefacts are stored in the repository. An artifact can be anything which can be stored as a file on a computer system, such as process models, software design documents, source code, executables, testing suites, *etc.*. Additionally the artefact includes *meta-data* which characterises it for the purposes of search and retrieval.

Networks As the OSCAR tool must operate in support of distributed teams, it need to allow and support communication.

It is the task of the OSCAR tool to manage artefacts. Users will interact with the tool to:

- Store artefacts,
- Retrieve artefacts,
- Update artefacts and
- Search for artefacts which match the user's needs (for example, to find a software component which implements a particular task).

As an example, the meta-data for an artefact could be:

Name	Value
Artefact name	Text entry widget
Creation date	25th December 2001
Creator	A Smith
Creator	J Bloggs
Modification date	14th January 2002
Modification comments	Fixed possible buffer-overflow
Modification date	26th January 2002
Modification comments	Now matches look-and-feel model for project

3 Example Use-Case

In this section, a simple use case [Jacobson et al., 1999] for the OSCAR tool is outlined and examined. The use case involves a user of the system (a software developer) retrieving a component from the OSCAR repository. In this example, the developer (who is the actor) provides a template for the artefacts which he or she is looking for. This template is then given to the local client to the OSCAR system which retrieves matching artefacts from the (distributed) repository. The template can, for example, be a simple set of keywords, in which case the search is similar to that executed by web search engines.

3.1 The 'Retrieve Artefact from Template' Use Case

As an outline, the sequence of events for this use case are as follows:

1. The Developer identifies him/herself to the system.
2. The Developer constructs a Template for the kind of artefact that they are interested in.
3. The Developer supplies the Template to the local client of the OSCAR system.
4. The local client searches the repository for artefacts which match.
5. The local client presents the results, ranking them in order of relevance to the template, to the user.
6. The Developer selects a matching artefact from those presented to him or her.
7. The local client retrieves the artefact and delivers it to the user.

Given that OSCAR is a distributed repository system, step 4 will involve the following:

- The local client performs a search on the all the meta-data available about artefacts in the distributed repository. For example, if there is a single centralised store of all the meta-data, the local client will query this store. If the meta-data is distributed throughout the network, each OSCAR repository will be searched.
- The meta-data repository or repositories search for items of meta-data which match the template.
- The meta-data repository or repositories return the matching meta-data.

Step 7 can be broken down into the following steps:

- The Developer asks the local client to fetch a particular artefact.
- The local client determines which of the distributed OSCAR servers holds the artefact in question.
- The local client requests that the particular server returns the artefact.
- The server transmits the artefact to the local client.

There are two alternative models mentioned above (centralised or distributed meta-data services). Figure 1 shows the networking connections required for a local client with a central meta-data server. Using a central server introduces a single point of failure into the system, but it avoids the problem of distributing meta-data throughout the system and maintaining consistency. This is a trade-off which will have to be examined more carefully before a decision can be made. In figure 1, the arc labelled 'search' corresponds to the connection made by the client to the repository during the search on the meta-data, while the arc labelled 'retrieve' corresponds to the connection made by the local client to the artifact repository which contains the particular artefact which the developer has requested.

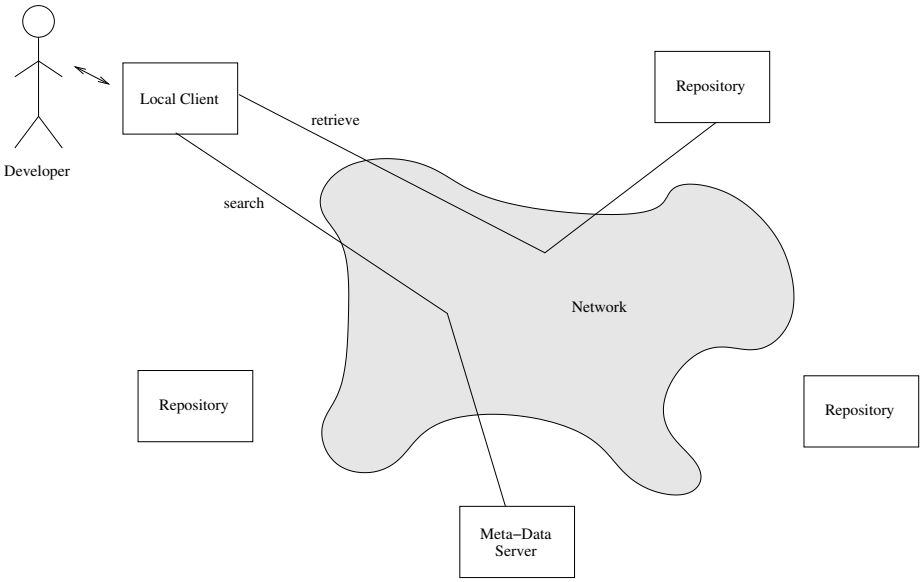

Figure 1: Searching for and Retrieving an Artefact

4 Requirements

4.1 Presentation Requirements

OSCAR must present the data it contains in a meaningful manner to clients which may be human, machine or a mixture of the two; for example a "guardian agent" approach where intelligent software mediates the human's interaction with the information. Consequently the following software requirements:

1. Intelligent data transformation. While OSCAR will have a common data format, clients and other external sources will have their own data that will need translating before it is meaningful, for both for input and output with OSCAR.

   Additionally the manner in which data is presented will change when the task the client is performing changes, and for human users with the experience level of the individual user.

2. Presentation of dependencies between artefacts. To facilitate tasks such as impact analysis the dependencies between artefacts of interest in OSCAR must be presented to clients as supplemental information during interaction. This will assist in alleviating the problems caused by changing superficially unrelated items.

3. Presentation of search results. Information generated by search jobs must be transformed into an appropriate form for the clients. While a printed report may be fine for a human they would be less useful to a tool looking for software components to compose.

4. Presence awareness. Like dependency presentation, the activities of other users should be indicated during interaction to facilitate collaboration.

Again, the means of indication must vary according to the characteristics of each client.

4.2 Indexing Requirements

OSCAR must index its contents to allow rich user queries. Queries are generally are intended to match appropriate artefacts to retrieve from OSCAR and also to score them at the client to present the best possible match.

5. Fuzzy/similarity based searching. Rather than just a simple boolean search like that found in many software products, the nature of OSCAR and the data contained within it requires a more sophisticated method of selecting search results. The type of search envisioned is conceptually similar to that outlined by Lai [Lai and Tait, 1998] that mimics the search techniques of humans looking for images.

6. Rich, extensible meta-data model to describe OSCAR's contents. Such a model is necessary to support the sophisticated searching and indexing OSCAR requires.

7. Reasonable response time. As the system is potentially highly distributed, searches must return a reasonable set of matches in a reasonable[1] time to ensure that a search does not take so long to complete that it impairs the performance of the rest of the system

8. Transparent distributed search. Whether an artefact (or its descriptive meta-data) is held locally or elsewhere in the system the search and subsequent retrieval of artefacts should be exactly the same.

4.3 Metrics Requirements

When users interact with OSCAR whether directly or from elsewhere in the GENESIS environment their actions naturally affect the environment. A record of these activities and their consequences provides an insight into the working practices of the virtual organisation built around OSCAR and GENESIS. Consequently a metrics system and extensible instrumentation package must be integrated with OSCAR to build this record. The requirements for this subsystem are as follows:

9. Transparent collection and calculation of metrics. To support the requirement for non-intrusive behaviour in GENESIS the metrics module should not require mandatory operation on the part of the user for correct operation.

10. Supplementing artefact meta-data with information derived from the metrics. Therefore the history and characteristics of the activities using an artefact should "surround" it when presented to the user and provide additional properties to search.

[1] "Reasonable" is defined by the client requesting the search.

11. Collation of certain metrics into additional documentation items. The information generated by some artefacts may not be useful when directly associated with the artefacts themselves. Instead a virtual document should be produced containing the information and associated with the artefact. This approach is intended to reduce information overload on human users.

12. Optionally extensible by users. If a user has a requirement for a metric not included in the standard instrumentation package, adding the metric should be simple.

4.4 Data Storage Requirements

The GENESIS system stores two kinds of data for each artefact, the *meta-data* and *artefact data*. Table 1 indicates the characteristics of each of these types:

	Meta-data	**Artefact Data**
Size	Small, relatively constant	Any size, variable
Structure	Highly structured	Partially structured
Operations	Mainly Query/Read	Read/Write balanced
Format	Plain text	Binary and plain text
Dependencies	None	Other artefact data
Links	Other meta-data	(optional) Data source

Table 1: Data Characteristics

The storage requirements for these two types of data differ significantly:

Meta-data
- Fast to search (sorted data)
- Lots of relational information
- Powerful query/edit system

Artefact Data
- Efficient storage (little wasted space or CPU time)
- Fast to access (no need to query a specialised program like a DBMS to retrieve data
- Error resistant (versioning,backoff etc)

This indicates the following software requirements:

13. A separate fast and structured store for meta-data, supporting a sophisticated query language.

14. A space efficient store for artefact data, containing a backup of any changes to meta-data.

15. A distribution model that allows optional distribution of both data stores in a *global name-space* while still retaining the error resistance and recovery properties of each.

4.5 Non-Functional and Inherited Requirements

The final set of requirements are those that are not directly related to OSCAR's functionality or are inherited from those of the wider GENESIS environment. Though they are not explicitly part of OSCAR they will still affect the design of the system:

16. The key requirement is that the GENESIS tool must be non-intrusive. GENESIS and consequently OSCAR must not gratuitously affect the existing organisational processes.

17. A high degree of dependability is necessary for OSCAR to be accepted and trusted with artefacts.

18. The requirement for distribution provides both dependability benefits (redundancy) and challenges (synchronisation and change control). The distribution models selected shall take both into account.

19. To facilitate the requirement for a non-invasive environment, GENESIS and OSCAR must be open and interoperable with existing systems.

5 Proposed Architecture

The sample use case for OSCAR provided earlier in the paper illustrates the four aspects of the artefact management system:

Presentation OSCAR shall be responsible for taking the stored content and delivering it to the user agents in the appropriate format. Some transformation or repackaging of data is expected here. In addition this layer shall implement the security (largely access control) and maintenance (expiring and compressing old data) features the repository requires.

Indexing The indexing system shall provide a method of searching and navigating the artefact repository's stored meta-data. The information provided by this layer forms the basis of the operations offered by the presentation layer which shall then act on the artefacts themselves.

Metrics The behaviour of the system must be captured for later study for purposes of quality assurance, supplementing the artefact meta-data, further user-needs analysis and research. The metrics engine will perform this task

Storage Large volumes of data must be stored efficiently by the system. The format of this data must be chosen to facilitate the following:

- Speed and ease of transformation at the presentation layer
- Speed and ease of indexing the data for retrieval

Raw size is not a problem as large artefacts may be distributed around the GENESIS repository network as necessary. Consequently a highly structured and extensible data format or set of data formats is necessary. Two types of data shall be stored *Meta-data* and *Artefact Data*. These

may be stored together or apart though if stored apart they must be linked in some way.

In figure 2 the two data stores are shown separately as most indexing operations shall be performed using meta-data only for speed.

The storage requirements of meta-data suggest that a DBMS might be appropriate. Relations between artefacts (linking and dependencies) may be encoded as part of the stored meta-data in the DBMS and in addition the DBMS's structured query language provides the fine-grained query and edit control necessary to effectively search the stored data. Finally the DBMS itself shall facilitate the query process by optimising the storage of the data for rapid querying.

Since many DBMS implementations support transaction services either internally or through a distributed transaction manager, this layer of the system is an ideal place to situate all synchronisation services for the rest of the GENESIS system.

Artefact data has rather different storage requirements; efficient storage and retrieval of specific data is of greater importance than search performance. Since the search capabilities of a DBMS are not required for this type of data a set of flat files on a standard file-system will give the greatest performance.

These two conflicting requirements suggest a two-tier architecture consisting of a DBMS to store meta-data to enable users to find artefacts and a set of flat files on disk to store the actual data. All data in the DBMS should also be stored in the flat files so that in the event of DBMS errors the repository can be repaired transparently from the data held in files. Similarly the DBMS's transaction services shall be used to prevent concurrency problems (many-writers) from corrupting the disk data store. This mirroring approach will not noticeably affect the performance of OSCAR since whenever the meta-data on disk is changed the (much larger) artefact data will also require modification.

5.1 Artefact Description Requirements

OSCAR shall represent all the information contained within it as a set of artefacts all of which possess properties, some common across all artefacts and others possessed by specialised artefact types. This will allow the repository to treat artefacts either as generic pieces of information (for high-level searches etc) or take advantage of additional operations for specialised artefact types.

5.1.1 Artefact Structure

Artefacts shall be structured recursively; at the lowest level every file is an artefact. Additionally, artefacts shall inherit properties from their parent artefacts: consider the example of an XML document with associated image and other data files. At the highest level the document will be represented by a single artefact containing several sub artefacts for each of the document components. Assuming the document was produced by a single actor the author and other such information will be inherited from the parent artefact. See figure 3 for a representation of such an artefact.

In light of this design decision there are some necessary constraints on the structure of artefacts:

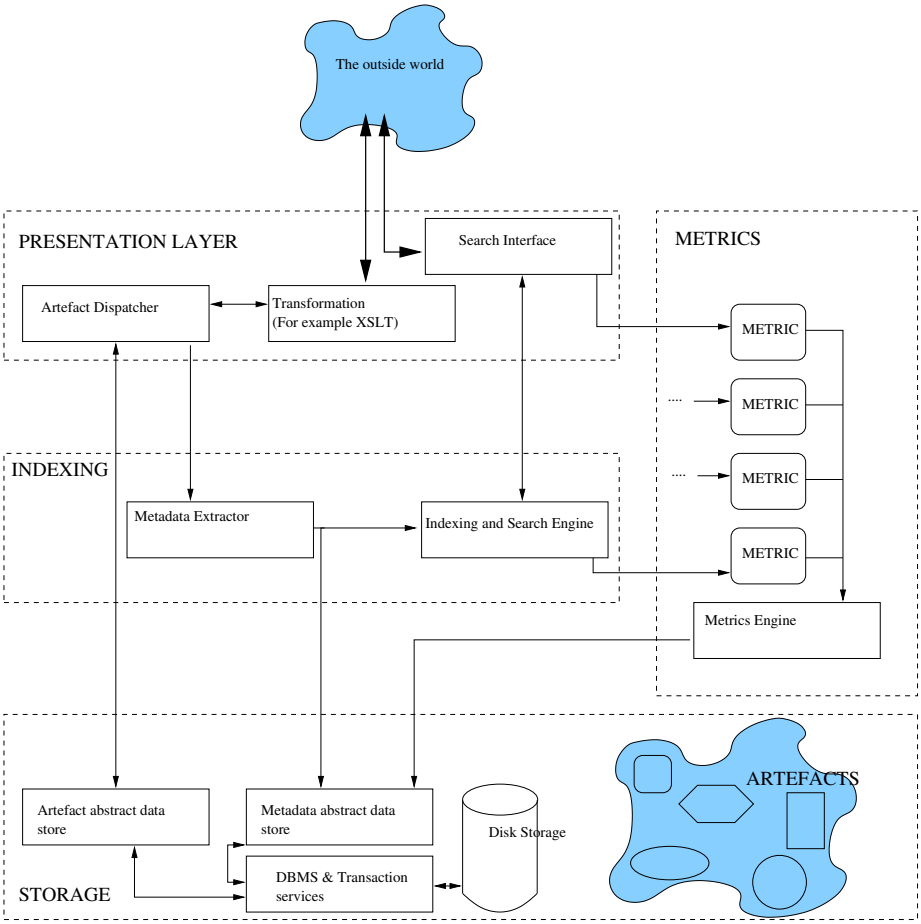

Figure 2: Overall Artefact Management System Architecture

- No two artefacts may "own" a single sub artefact. Put another way, artefacts may not inherit properties from more than one artefact. This decision has been borrowed from the design of the Java programming language and is intended to avoid the so called "Diamond Of Death" caused by conflicting properties. Though dual-ownership of an artefact is not possible, artefacts may be re-used by linking from another artefact and the appropriate construction placed upon the relationship when the artefact is presented.

- Explicit properties assigned to sub-artefacts override the corresponding properties in their parent artefact.

Aside from inheritance of properties from another artefact and encapsulation of sub-artefacts, artefacts also possess linking relationships with other artefacts. Consider our XML document and its associated images again. A single image (for example a chart) within the document may be generated by some process described elsewhere in the repository and though the image does not inherit properties from this process it *depends* upon that process to exist at all. Several

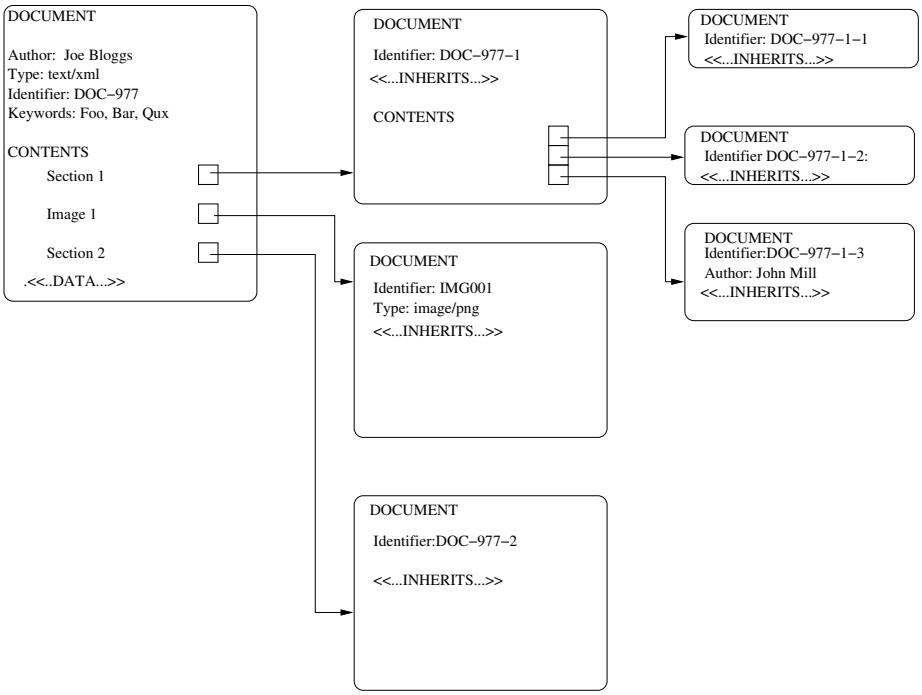

Figure 3: Structure Of An Example Artefact

types of linking relationship shall exist within the system:

Link Used to indicate artefacts associated with another by some informal link

Describes Used to indicate another artefact which documents the current one.

Suggested Indicates an artefact that is strongly associated with the current one but is not required to retain the current artefact. An example of such a relationship would be an artefact representing code for a client suggesting the related server-side software

Dependency Used to indicate an artefact that the current one depends upon

This resembles the Debian [Jackson and Schwarz, 1998] configuration management system for packaged software. Unlike the inheritance capabilities described above cyclic relationships are possible and must be detected and eliminated.

5.1.2 Data Formats

Any potential technologies for describing artefacts must satisfy several important properties:

Lightweight The barrier to entry for the technology (in terms of TCO, setup complexity etc) must be low enough to ensure OSCAR can be deployed quickly and easily.

Extensible Artefact descriptions including specialised artefacts should be extensible in a backwards compatible way at any time by the user

Open/Free and interoperable Since non-intrusivity is a requirement of the GENESIS system as a whole, OSCAR must employ interoperable technologies with excellent tool support to ease integration with existing systems.

Considering that OSCAR will already possess some form of DBMS to provide transaction services and store meta-data one potential technology is a relational database. In particular the entity relationship model supported by these databases will provide a powerful linking mechanism the integrity of which is enforced by the DBMS. Though the database schema is extensible on-the-fly without affecting data integrity, each specialised artefact type will usually require a new relation within the database. Finally not all RDBMSs deal well with large quantities of text data.

XML [Bray et al., 2000] offers another potential technology for artefact description. In particular its modular construction allows features from a variety of other XML document types to be imported with minimal effort. Additionally, XML-Schema provides a validation system equal to that provided by many databases. Finally XML is an extremely portable technology with excellent tool support across all platforms, easing the interoperability burden of the system.

5.1.3 Generic Meta-Data

All artefacts will possess *meta-data* to aid indexing and searching. As described earlier each artefact will have some generic properties and some specialised properties according to its role. The generic meta-data included/inherited by each artefact shall include:

- A unique identifier
- A descriptive title
- Original creator/actor with primary responsibility for the artefact
- Contributing authors
- Date created/modified
- Validity over time
- Link relationships, such as dependencies and reuse related links
- Subject keywords
- Information on the process used to create the artefact (if any)
- Access permissions.
- Versioning information

Additionally, change information and other logs shall be treated as meta-data wherever possible. This will provide a notion of presence as "footsteps in the sand"

Most of this generic meta-data can be structured at the presentation level using a specialised version of an extensible meta-data vocabulary. The necessary schema may be imported as a name-space into the schema or DTD used to structure the artefact descriptors.

5.2 Dynamically Created Meta-Data Requirements

This type of "documentation" within the GENESIS system is generated on demand and associated with its source artefacts. It shall be used to store items of information that are not necessary for a software artefact to be used in a software process but provide useful perspective or helpful secondary information. Such artefacts shall consist of:

- API/technical documentation of software,
- User comments and other informal documents,
- Change logs,
- Metrics output such as the level of interaction by other users with the target artefact and
- Organisational knowledge such as interesting references, best practice recommendations, anecdotes etc.

5.2.1 Styling

At the presentation layer the contents of the repository shall be transformed into information appropriate for the various users of the system whether machine (agent) or human. Since the data will be initially presented as XML, XSL and an appropriate transform language is appropriate to conduct this activity.

The behaviour of OSCAR at this level resembles existing document/content management systems that apply a stylesheet to content before transmitting it to (usually web-based) clients. Ideally styling and transformations should be entirely transparent to the user though since OSCAR cannot know all a user's requirements beforehand a facility should be provided to indicate to the system exactly what a user requires.

5.3 Indexing and Retrieval Requirements

OSCAR shall permit clients to search the system to retrieve artefacts that match their needs. Two methods to do this shall be presented to the user, both relying on one underlying search technology:

1. An Internet search-engine style interface permitting searching by keyword, boolean connectives and ordering of results by a variety of metrics. As most users are familiar with Internet search engines this interface will flatten the system learning curve.

2. An interface which allows an artefact descriptor to be passed in as a *comparator* Artefacts that match the comparator shall be returned The other search interface shall generate a comparator encompassing the search properties.

6 Proposed Classification Of Artefacts

Though each artefact may be treated as a generic object, instances of artefacts will be of specialised types with a variety of properties.

6.1 Software

The software artefact will represent both requirements and design artefacts, for example the information generated by CASE tools as well as source code for software modules. The *Software* artefacts should possess the following additional meta-data:

Tools The tools used to create the artefact (as references to a *Tool* artefact).

Re-use Information Any reuse specific information associated with the artefact. In particular, references to the *ProcessInstances* where this artefact is used and information relating to the composability of artefacts as reusable software components should be here.

Services Required Other software artefacts that the artefact depends on. Effort should be made here to prevent cyclical dependencies as outlined in section 5.1.1

Services Provided In the context of software code, this includes interface details for the module or component. In the context of a design or requirements software artefact this attribute should be null.

Both the Services attributes should wherever possible be described using standard Interface Description Languages such as the Web Services Description Language [Christensen et al., 2001].

6.2 Documentation

Documentation artefacts in OSCAR are any items of supplementary information added by users or automatically generated by internal OSCAR processes. They are not intended output from any software process but rather incidental information produced while a process is being enacted. All components of documents such as images should also be classified as *Documentation*.

Any documents produced as part of a formal software process such as Requirements or Design documents shall be designated a *Software* artefact.

Documentation generated by OSCAR's internal systems (such as the metrics engine) produce a special type of *Documentation* artefact known as a *Journal*. See section 6.8 for more information on this artefact type.

Documentation shall be stored as XML for which stylesheets and formatting objects exist to transform the content into distribution formats such as PDF or HTML.

6.3 Tool

The *Tool* artefacts describe external programs that may be executed and may optionally act upon a set of external artefacts. Behaviours described by the tool artefact may include the following:

- Allows the invocation of a third-party tool by GENESIS to produce OSCAR artefacts. The *Tool* artefact takes care of translating the tool's input and output via the presentation layer.
- Describes a metric. "Executing" the metric produces the current results. In reality of course the metrics will be calculated behind the scenes by the metrics engine.

To permit these behaviours the tool artefact must possess the following properties:

Invocation information Details of how the tool should be invoked by GENESIS clients. This allows the addition of a tool-server system like that employed by Oz at a later date and the interfacing of OSCAR to external software repositories.

Services required A list of other *Tool* identifiers that this *Tool* requires to perform its tasks.

Service provided An identifier describing the service the tools provide. For example, `emacs`, `vim` and `pfe` would all be examples of tools providing the `TextEditor` service.

The service keywords used in each system should be specified at system deployment or allowed to evolve in an ad-hoc fashion. Generally of course artefact authors will not specify abstract tools but the specific tools they used to create an artefact.

Target types An optional list of artefact types that the tool may act upon

Output types An optional list of artefact types that the tool produces

6.4 Actor/User

The *Actor* (also synonymous with *User*) artefact type in OSCAR is necessary for two major parts of the system:

Security Without access to a valid *Actor* artefact users of the system will not be able to perform any activity. This ensures that all actions (even simple reads by anonymous users) may be tracked using the metrics engine.

Author/Role Information Actor artefacts are used to fulfil the creator attributes of all artefacts and the role attributes of *ProcessElement* and *ProcessInstance* artefacts.

To facilitate both of these activities the *Actor* must possess the following properties:

- Whether the *Actor* is human or machine. Machine actors (such as agents) differ from *Tool* artefacts in that the former act externally from GENESIS whilst the latter are invoked by GENESIS to perform a specific tasks.
- Authentication information.
- Permissions information.
- Contact details (for human actors).

6.5 Legacy

The *Legacy* artefact is essentially a plain artefact which encapsulates a quantity of legacy data that is not stored in a way that OSCAR can understand or index correctly. For example, a large set of documents in proprietary formats that have not been managed as *Documentation* artefacts from the outset may be treated as *Legacy* artefacts until such time as they can be entered as proper artefacts.

Consequently the only additional attribute of a *Legacy* is a date to indicate when it is planned that the contents of the Legacy artefact will be added as true artefacts. Absence of the date indicates the Legacy will not be integrated.

6.6 ProcessElement

A *ProcessElement* describes part of a work-flow process. Like other artefacts they are recursively defined, allowing easy re-use of any appropriate subset of a process. The process itself shall be described by the use of a formal or semi-formal Work-flow Definition Language (WDL)

A *ProcessElement* shall have the following additional properties to support these tasks:

Inputs The artefact types that the process will consume (if any) when run.

Outputs The artefact types that the process will produce (if any).

Roles The *Actors* involved in executing this particular work-flow step

Prerequisites The pre-requisites for this step to execute successfully, such as other processes completing successfully etc.

A process element itself cannot be executed to implement a particular work-flow. Instead, the top level *ProcessElement* must be *instantiated* as a *ProcessInstance* which is linked to a specific version of the *ProcessElement*. Therefore, abstract *ProcessElements* can be used to define re-usable processes whilst concrete *ProcessInstances* can be used to record the execution of each process, providing useful information for future process design and analysis of working practices.

6.7 ProcessInstance

Once a *ProcessElement* is instantiated, all its sub-elements (if any) are also instantiated as *ProcessInstance* artefacts, preserving the relationship between them. To differentiate multiple instances of a running process from their template *ProcessElement* a *ProcessInstance* must have the following additional attributes:

Instance Identifier For all sub-instances this attribute shall denote the top level *ProcessInstance* artefact in the running process.

Element Version The version of the template *ProcessElement* that spawned this process. This is to ensure that if the best-practice process changes in the future completed *ProcessInstances* may be studied in the context of the original process

State The current state of this instance:

- Ready To Run
- Running
- Successful completion
- Unsuccessful completion

6.8 Journal

The *Journal* artefact represents versioning/change information for each artefact in the system. Each artefact will have an associated *Journal* to "store"[2] this information and present it as a coherent document. In particular the *Journal* will be used as a means of conveniently packaging the output from metrics *Tools* applied to the artefact. Since *Journal* artefacts are virtual, they possess no additional attributes. The metrics and other information in each *Journal* depend on the base artefact type.

For this reason Journal artefacts cannot be recursively defined like most other artefact types. Should a more concrete representation of a *Journal* need to be stored in the repository, the results of examining a Journal at a particular time may be saved as a *Documentation* artefact.

7 Implementation and Exploitation Strategy

Open-source software is typically developed incrementally [Raymond, 1999]. In the GENESIS project, the plan is to develop the software in a closed-source style for the first year, and to release the software into the open at that point. From that point onwards, development will take place in the open. Initially (during the first year), the development will follow a traditional closed-source lifecycle, which is currently at the stage of gathering and analysing requirements.

[2] In reality this information will be generated automatically when the artefact is retrieved since storing such information in discrete artefacts is unlikely to be efficient in space nor easy to keep current.

7.1 Exploitation Strategy

There are two categories of members of the GENESIS consortium: universities and companies. The universities will use the results of the project for teaching and research purposes (publishing papers, *etc.*, while the companies will exploit the results by using the tools and expertise generated during the project. These results will enable them to improve their software engineering processes using both the tools (*e.g.*, OSCAR) and the methods for workflow and configuration management.

8 Conclusions

The motivation and requirements for OSCAR, an Open Source software repository, have been described and discussed. OSCAR is a part of the GENESIS project, an initiative to develop an enterprise class software engineering environment for distributed collaborative working. The next phase is to complete the design of the entire GENESIS system in collaboration with our European partners. Further information may be found at `http://www.genesis-ist.org`.

The key requirement for GENESIS as a whole is that it is lightweight and non-intrusive and therefore will not disrupt existing successful organisational practices and furthermore may be adapted to facilitate those practices in any organisational context. OSCAR is intended to provide a distributed and dependable repository with a global name-space for managing all artefacts related to software engineering including process models, source code and other kinds of documents.

We have presented a proposed architecture for such a system as represented in figure 2. The architecture is intended to facilitate active artefact management, a collective term for several automated facilities including change traceability and logging, dependency analysis, dynamic artefact data transformation and history/presence as meta-data. The heart of this technique will be in OSCAR's indexing and metrics components.

References

[Bray et al., 2000] Bray, T., Paoli, J., Sperberg-McQueen, C., and Maler, E. (2000). Extensible Markup Language (XML) 1.0 (second edition). Technical report, The World Wide Web Consortium.

[Christensen et al., 2001] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., and Research, I. (2001). Web services description language version 1.1. Technical report, The World Wide Web Consortium.

[Flores et al., 1988] Flores, F., Graves, M., Hartfield, B., and y Winograd, T. (1988). Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6(2):153–172.

[Gail E. Kaiser and Yang, 1997] Gail E. Kaiser, Stephen E. Dossick, W. J. and Yang, J. J. (1997). An architecture for WWW-based hypercode environments. In *1997 International Conference on Software Engineering: Pulling Together*, pages 3–13, Boston MA.

[Haake, 1999] Haake, J. M. (1999). Openess in shared hypermedia workspaces: The case for collaborative open hypermedia systems. *ACM SigWEB Newsletter*, 8(3):33–45.

[Jackson and Schwarz, 1998] Jackson, I. and Schwarz, C. (1998). The Debian policy manual. Technical report, The Debian Project.

[Jacobson et al., 1999] Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley.

[Jiang et al., 1997] Jiang, W., Kaiser, G. E., Yang, J. J., and en E. Dossick, S. (1997). Webcity: A WWW-based hypermedia environment for software development. In *7th Workshop on Information Technologies and Systems*, pages 241–245.

[Kaiser, 1998] Kaiser, G. E. (1998). WWW based collaboration environments with distributed tool ser vices. *World Wide Web Journal*, 1(1):3–25.

[Lai and Tait, 1998] Lai, T.-S. and Tait, J. (1998). General photographic image retrieval simulating human visual perception. In *Proceedings of the ACM SIGIR'98 Post-Conference Workshop on Multimedia Indexing and Retrieval*, pages 17–28, Melbourne, Australia.

[Leblang, 1994] Leblang, D. B. (1994). The CM challenge: Configuration management that works. In Tichy, W. F., editor, *Configuration Management*, Trends In Software, chapter 1, pages 1–37. John Wiley and Son.

[Oinas-Kukkonen and Rossi, 1999] Oinas-Kukkonen, H. and Rossi, G. (1999). On two approaches to software repositories and hypertext functionality. *Journal Of Digital Information*, 1(4).

[Raymond, 1999] Raymond, E. S. (1999). *The Cathedral and the Bazaar: Musing on Linux and Open Source by an accidental revolutionary*. O'Reilly and associates.

[Sachs, 1995] Sachs, P. (1995). Transforming work: Collaboration, learning, and design. *Communications Of The ACM*, 38(9):36–44.

[Selvin, 1999] Selvin, A. M. (1999). Supporting collaborative analysis and design with hypertext functionality. *Journal Of Digital Information*, 1(4).

A Glossary

Artefact A product, whether formal or informal from the process of software engineering.

DBMS Database Management System. Various types exist.

Free Software A more rigorous type of Open Source, with the intention of producing of "Free Speech" (rather than "Free beer") software.

GENESIS Generalised Environment for process management in cooperative software engineering.

Meta-data In the context of OSCAR supplementary data stored for the purpose of indexing and searching the repository.

OSCAR Open Source Component Artefact Repository

OSS Open Source Software. Specifically software that complies with the Open Source Definition.

SCM Software Configuration Management

TCO Total Cost Of Ownership

WDL Work-flow Description Language. An example is the Web Services Description Language [Christensen et al., 2001]